ORACLE
INSURANCE

Oracle® Documaker

Programmer's Guide to Docusave Server

version 3.0

Part number: E15145-01

September 2006

THIRD PARTY SOFTWARE NOTICES

This product includes software developed by Apache Software Foundation (http://www.apache.org/).
THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software distributed via the Berkeley Software Distribution (BSD) and licensed for binary distribution under the Generic BSD license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software developed by Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

This product includes software developed by the Massachusetts Institute of Technology (MIT).
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This software is based in part on the work of the Independent JPEG Group (http://www.ijg.org/).

This product includes software developed by Sam Leffler of Silicon Graphics.
THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE

This product includes software components distributed by Steve Souza.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2002, Steve Souza (admin@jamonapi.com). All Rights Reserved.

This product includes software components distributed via the Berkeley Software Distribution (BSD).

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2000-2006 www.hamcrest.org. All Rights Reserved.

This product includes software components developed by the Independent JPEG Group and licensed for binary distribution under the Independent JPEG Group license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1994-1998 AIIM International. All Rights Reserved.

This product includes software components developed by Sam Stephenson.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2005-2007 Sam Stephenson

This product includes software components developed by Sun Microsystems.

Copyright (c) 1995-2008 Sun Microsystems, Inc. All rights reserved.

This product includes software components distributed by Vbnet and Randy Birch.

Copyright © 1996-2008 Vbnet and Randy Birch. All Rights Reserved

---

This product includes software components distributed by MetaStuff.

THE SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS-IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDINGBUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS AND SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, ARISING IN ANY WAY OUT OF THE USE OF THE SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software components distributed by JSON.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This product includes software components distributed by OpenSSL (http://www.openssl.org/).

THIS SOFTWARE IS PROVIDED BY THE OPENSSL PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENSSL PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA OR PROFITS; OR BUSINESS INTERRUPTION) HOWVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

This product includes software components distributed by Yahoo! Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Publication history

First issue for /NT 2.0.4: July 2002
Revision 1 for /NT v2.0.8/6000 v2.0.4a GA: September 2003
Reissued for /NT v3.0: May 2006
Revision 2: for v3.0 running on Windows, AIX, Solaris and Linux: September 2006

Table of Contents

Table of Contents

Docusave Server documentation roadmap

Administrating

Installation Guide

Server Administrator Guide

Server Help

Programming

Preface

Welcome

Docusave for Oracle offers a comprehensive range of scalable high-performance products for every step in the life cycle of a document. These include Docusave for Oracle Creation Solutions to capture data and create forms, Oracle Publishing Solutions to produce large volumes of personalized documents, **Oracle Archival Solutions** to intelligently store and retrieve documents, Oracle Management Solutions to control and network documents, and Oracle Development Tools to customize your Oracle Solutions implementations and interfaces.

Docusave Archival Solutions store your documents electronically for intelligent retrieval and viewing. Docusave Archival Solutions facilitate immediate access to your documents for such applications as claims processing, enterprise-wide contract or regulatory document lookup, call center statement reference, and other internal and customer service-based functions. You can implement Docusave Archival Solutions as standalone systems or integrate them with leading imaging systems.

A key component of Docusave Archival Solutions is the Docusave Server™ application.

Using this manual

This *Programmer's Guide to Docusave Server* manual is written to help you customize or extend base product functionality.

Conventions

The *Programmer's Guide to Docusave Server* manual provides consistent typographic conventions and keyboard formats to help you locate and interpret information easily. These conventions are provided below.

Convention	Description
Italics	Command, dialog box, button, and field names
Arial font	Directory, folder, and file names
1 **Numbered lists**	Provide step-by-step procedures for performing an action
◆ Bulleted lists	Provide grouped information, not procedural steps

Related documents

In addition to this Programmer's Guide, the related documentation described here is also included with Docusave Server. This documentation includes administrator documentation and programming documentation.

Installation Guide

The *Installation Guide* is written for administrators of the Docusave Server application. It describes how to install the Docusave Server application on operating systems that it is compatible with.

Administrator's Guide

This *Administering Docusave Server* guide is written for administrators of the Docusave Server application. This guide describes how to use Docusave Server on multiple operating systems to automate document filing and document batch retrieval.

Docusave Server Help

The *Docusave Server Help* provides an overview of Docusave Server. It also describes the Job Recognition, Processing and Imaging Libraries, the BldCache and Addres utilities, and the AddResources batch file.

Docucorp Queue Systems Guide

The *Docutoolbox Docucorp Queue Systems* guide shows you how to install, configure, and manage the Queue Management Systems. Queues provide a

way for other applications to organize jobs for processing in Docusave Server. Docusave Server uses queues as holding areas for jobs awaiting processing. Dedicated queues and file servers improve the throughput processing time for Docusave Server by distributing jobs across multiple computers available on a local area network.

What is Docusave Server?

Introduction

Docusave Server automates document filing and batch retrieval.

When performing document filing, Docusave Server processes jobs, which consist of groups of documents, separates them into individual documents, converts and optionally compresses each document, and then automatically files the documents into an archival imaging or document management system.

When performing batch retrieval, Docusave Server fetches previously archived documents for subsequent processing by another process (like Docucreate IC) or for routing print stream documents back to a production printer. It does this by using two types of records:

- Document RePrint records (DRP records)—Docusave Server retrieves a compressed AFP or Metacode print stream and its dependant printer resources from the archival imaging or document management system for subsequent printing.

- Document Retrieval Requests (DRR records)—Docusave Server retrieves page images from an archival imaging or document management system and exports them in a format accepted by Docucreate IC.

Use and Structure

Docusave Server can be used as part of any business solution that requires automated processing (typically archiving or batch retrieving) of significant volumes of documents or document oriented transactions. Additionally, Docusave Server provides transaction level logging, error recovery, and custom library support to meet unusual or changing business needs.

The custom capabilities of Docusave Server enable novel solutions and allow easy development of:

- Custom output destinations (e.g., new imaging or archival system destinations)
- Custom index processing including validation, “pre-flighting,” and/or possibly correcting indexes before a document is filed
- Custom pre-, post-, and processing steps, such as file type conversion, data validation, and error recovery

Docusave Server takes its primary input from a queue. Use of a queue allows business solutions to span separate computing platforms, where the applications that create documents reside on a different computer system (or even a different type of system) than Docusave Server itself. Another advantage of queues is that the separate processes feeding a queue can be started and stopped independently of Docusave Server. Using queues also allows the number of instances of Docusave Server to differ from the number of processes feeding the queue, which permits load balancing and system scaling. In short, use of an input queue provides Docusave Server a single interchange point for improved platform connectivity, independent operation, load balancing, and system scaling.

The elements that reside in a Docusave Server input queue are groups of documents or transactions in a single unit, called a job. A job can contain one

or a thousand documents. One important function of Docusave Server is to split jobs into separate documents for individual processing, and to ensure that the job process is complete. An axiom of Docusave Server design is that entire jobs either succeed or fail. Error recovery processing is provided to ensure that a job does not partially succeed.

The “all or none” approach to document processing within a job provides important operational simplicity. It eliminates messy recovery in situations where part of a job was successfully processed, and part of it failed. “Roll-back” logic is included in Docusave Server so that jobs either succeed entirely, or the system is rolled back to its state before the job was started. (If some documents were processed in a job that fails, roll-back ensures the documents are deleted from the output destination).

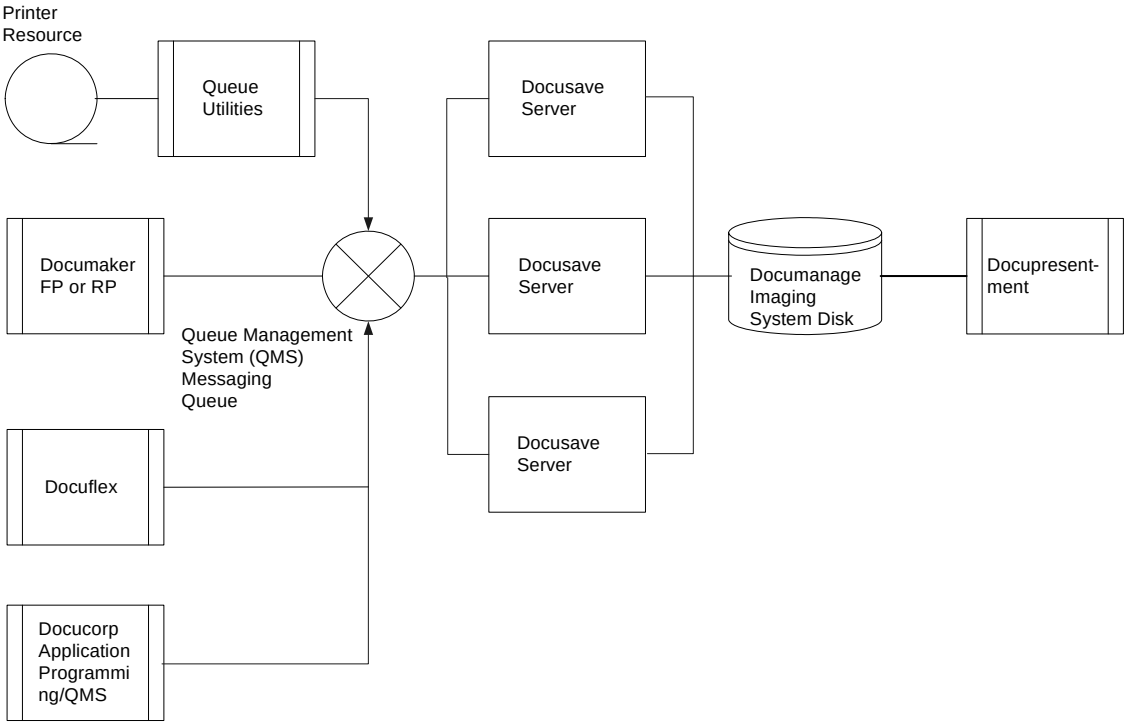
If documents are intended to be processed independently, and not as a combined job, they can be queued in separate jobs of one document. (Note: Some queue implementations have performance implications or system limitations on the number of jobs in a queue. Keep this in mind when putting many small jobs in a queue.).

Docusave Server in Context

In a filing scenario, Docusave Server is often used in conjunction with Documaker or Docuflex publishing solutions. Typically the output of Documaker or Docuflex is directed to a Docusave Server input queue. Docusave Server in turn often writes its output to an archive or document management system such as Documanage.

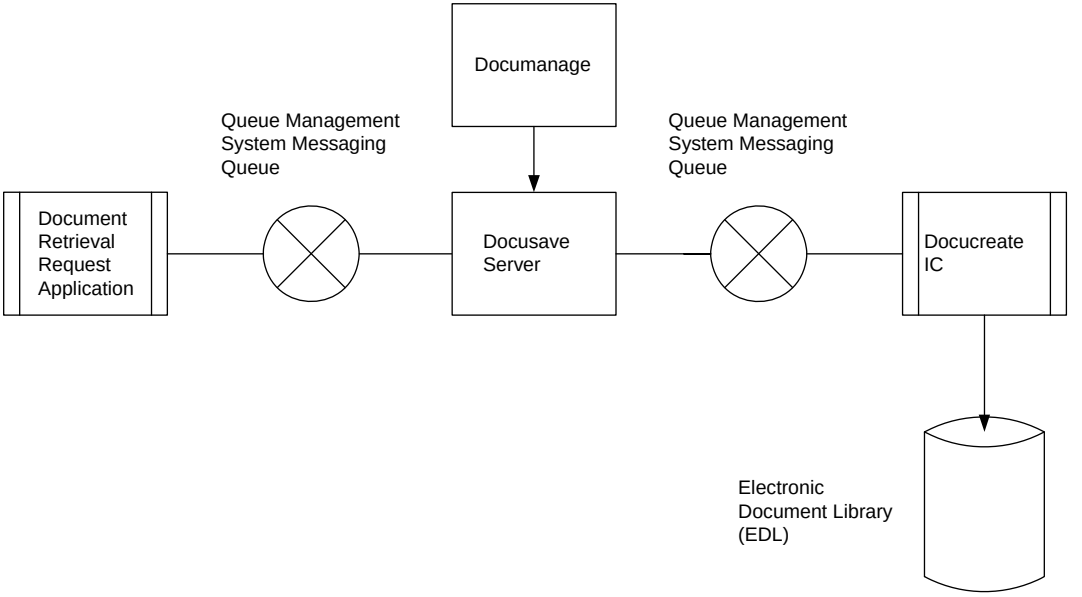
Together, Documaker/Docuflex, Docusave Server, Documanage, and Docupresentment, with their associated tools, form the architectural foundation for a number of complete business solutions.

Docusave Server can be part of a system designed to process very high data volumes. It may be desirable to run multiple instances of Docusave Server concurrently for increased throughput. Typically, multiple publishing processes feed a single input queue from which multiple instances of Docusave Server process jobs in parallel.



In a batch retrieval scenario using DRR records, Docusave Server runs “in reverse” to extract page images from an imaging or document management

system and emits them as “megafiles” into a queue for processing by Docucreate IC.



Flow of Control

Docusave Server follows the same sequence of processing steps regardless of variations in processing content and configuration. For example, the file type translation, compression choice, and destination archive or imaging system can all be changed without changing the Docusave Server program, its internal structure, or process flow.

Within the general flow, there are a number of places where Docusave Server offers processing options. Some options are controlled by a user interface (or

the equivalent configuration files or input control statements). Other processing options are determined by library selection, which can be static or dynamic.

Dynamic library selection is governed by two attributes of the job being processed. A job recognition library examines a job in the input queue and returns two processing types: the Job Type and Process Type (also known as the Input type and Output type, respectively). These two types are represented with three letter codes, which trigger the naming conventions for processing libraries. For example, 'RAS' designates rasterization, and 'PRT' designates print stream format. The job and process type are used to dynamically select and load the libraries for job and document processing steps.

Summarized, the internal processing flow of Docusave Server is as follows:

- Initialize and begin monitoring an input queue for work units, called "jobs"
- Wait for a job in the input queue
- For each job in the input queue:
  - Call a "job recognition" library to determine the processing requirements, and set the corresponding Job Type and Process Type
  - Call the job pre-processor function (if any)
  - Call the job processor function which bursts the job into separate documents (transactions). If the job contains documents that need to be processed individually, the job processor will call the following document processing libraries
    - Document pre-processing library (if any)
    - Document processing library
    - Document post-processing library (if any)
  - Call the job post-processor library (if any)

- Start the next job in the input queue, or resume waiting for more
- Upon exiting, release memory, network, and system resources, close files, and return to the operating system

The normal inputs and outputs to these process are:

- Input queue (into which jobs are placed by other processes)
- Configuration settings, gathered from a User Interface in the Windows version, otherwise from control card settings or configuration files
- Output to log files that show jobs and documents successfully processed, and their disposition
- Output to log files that show jobs and/or documents which failed to process, the reason for failure, and the action taken or recommended
- Output to the chosen processing destination, usually an archival imaging or document management system, but sometimes simply as files on disk
- Megafiles output to a queue, formatted for processing by Docucreate IC.
- Reprint packages output to disk, formatted for subsequent printing.
- Output and Input among various temporary files during processing

Although Docusave Server can be configured many different ways, certain configurations using the default libraries cause it to perform the specific functions of previous products:

- AccessCommander functionality can be achieved by using Docusave Server to process AFP or Metacode print streams, or stacked DCD documents. This processing includes bursting jobs into separate documents, analyzing resource requirements, ensuring required resources

are accessible, performing DJDE normalization (if applicable), and archiving to a supported imaging system. In addition to documents, printer resources can be archived using Docusave Server, fulfilling another AccessCommander function.

- ImageCreate functionality converts AFP or Metacode print streams into TIFF (or similar) image files before archiving those into an imaging system.

Other configurations are possible when used with custom libraries and provide virtually unlimited flexibility in processing.

Library Overview

NOTE: All libraries are supported by all operating systems (Microsoft Windows, AIX, Linux and Solaris) unless otherwise specified.

When Docusave Server is started it loads and initializes the following libraries:

- DSJOBTYP – for recognizing the contents of input jobs
- CODECMGR – for managing the compression of documents and the decompression of input jobs
- DSISYSxx – for storing/retrieving documents to/from the imaging system. Each imaging system will have its own library, and the standard installation includes support for:
  - DSISYSD - Disk (local file system)
  - DSISYSDM - Documanage

- DSISYSFL - Output File
- DSISYSFN - FileNet: Windows, AIX
- DSISYSIC - ImageCreate Output: Windows, AIX
- DSISYSM - MARS/NT: Windows
- DSISYSV - VLAM EDL: Windows

Docusave Server is often used in conjunction with a publishing solution such as Documaker or Docuflex. The output of Documaker or Docuflex is directed to a Docusave Server input queue. When a job is extracted from the input queue the job recognition library is called to identify the job type. It returns a three-character string that uniquely identifies the contents of the job. Docusave Server will then use this string as a suffix to construct the names of the job pre-processing library, the job processing library, and the job post-processing library. The names will have following format:

- DSPRJxxx – pre-process job
- DSPJBxxx – process job
- DSPOJxxx – post-process job

For example, if the job-typing library returns the string "**ABC**," then Docusave Server would attempt to load the three libraries named

- DSPRJ**ABC**
- DSPJB**ABC**, and
- DSPOJ**ABC**

The job processing library is the only library that must exist. The pre/post-job/document processing libraries are optional and are not supplied with the standard installation; they are custom libraries.

The following job processing libraries are supplied with a standard installation:

- DSPJB**DCD** – for processing DCD input
- DSPJB**DRP** – for processing DRP input

- DSPJB**DRR** – for processing DRR input
- DSPJB**FPP** – for processing DMFPPP logs
- DSPJB**IMC** – for processing ImageCreate output
- DSPJB**PCL** – for processing PCL input
- DSPJB**PRT** – for processing AFP and Metacode input
- DSPJB**RSC** – for processing printer resources

A job processing library calls document processing libraries if the job contains documents that need to be processed individually. The following document processing libraries will be supplied with a standard installation:

- DSPDC**DCD** – for processing DCD documents
- DSPDC**NUL** - for transferring AFP/Metacode documents to a mainframe VLAM library as-is
- DSPDC**PCL** - for archiving PCL documents
- DSPDC**PRT** – for archiving AFP/Metacode documents
- DSPDC**RAS** – for rasterizing AFP/Metacode documents

Once jobs are processed, they are output to an imaging system and Docusave Server begins processing the next job in the queue or resumes waiting for more.

Sample Code

Docusave Server is shipped with sample C++ code to illustrate the implementation of each type of library used. This is an excellent starting point for custom-built libraries.

- **DSJBTXYZ.** A sample job recognition library. This library doesn't actually interrogate the content of jobs but always returns “XYZ” as the job type, which triggers the other sample job processing libraries.
- **DSPRJXYZ.** A sample job pre-processing library. This library does not alter the contents of the job but writes messages to the server log to indicate that it has been invoked.
- **DSPJBXYZ.** A sample job processing library. This library loads and invokes the *XYZ document processing libraries. For the purpose of this sample the entire job is treated as a single document.
- **DSPRDXYZ.** A sample document pre-processing library. This library does not alter the contents of the document but writes messages to the server log to indicate that it has been invoked.
- **DSPDCXYZ.** A sample document processing library. This library sends the entire document to the imaging library for storage and intentionally returns an error code to demonstrate that the document post-processor can alter this return code.
- **DSPODXYZ.** A sample document post-processing library. This library reduces the return code from the document processor to a maximum of four (4), which is intended to simulate reducing an error condition to a warning condition.
- **DSPOJXYZ.** A sample job post-processing library. This library merely writes messages to the server log to indicate that it has been invoked.

- **DSISYSU.** A sample imaging library. This library merely writes messages to the server log to indicate that it has been invoked.

Common Elements

Introduction

Docusave Server takes its primary input from a queue. Use of a queue allows business solutions to span separate computing platforms, where the applications that create documents reside on a different computer system (or even a different type of system) than Docusave Server itself. Another advantage of queues is that the separate processes feeding a queue can be started and stopped independently of Docusave Server. Using queues also allows the number of instances of Docusave Server to differ from the number of processes feeding the queue, which permits load balancing and system scaling. In short, use of an input queue provides Docusave Server a single interchange point for improved platform connectivity, independent operation, load balancing, and system scaling.

The elements that reside in a Docusave Server input queue are groups of documents or transactions in a single unit, called a job. A job can contain one or a thousand documents. One important function of Docusave Server is to split jobs into separate documents for individual processing, and to ensure that the job process is complete. An axiom of Docusave Server design is that entire jobs either succeed or fail. Error recovery processing is provided to ensure that a job does not partially succeed.

The “all or none” approach to document processing within a job provides important operational simplicity. It eliminates messy recovery in situations where part of a job was successfully processed, and part of it failed. “Roll-back” logic is included in Docusave Server so that jobs either succeed entirely, or the system is rolled back to its state before the job was started. (If

some documents were processed in a job that fails, roll-back ensures the documents are deleted from the output destination).

Pre-initialization functions

Some of the libraries contain functions that may be called before the initialization function. Currently there are two types of functions that fall into this category:

- **Identity functions.** Return a character string that provides a brief description of the library's function. The server can use this information to display in dialogs and in logged messages.

- **Setup functions.** In a Windows environment this function is called by the server to display any setup dialog that might be appropriate for the library.

Initialization functions

Each of the libraries contains an initialization function, which is normally called before any other function.

Typically the initialization function takes at least two parameters:

- The address of a four-byte control block address

  The control block address will be set by the initialization function and will subsequently be passed to all the other functions.

- The interface version

  Each header contains a version definition which can be used to indicate the interface version. For example, the job processing interface “dsprocj.h” contains the definition DSPROCJ_VERSION which should be passed as the value for the interface version.

Termination functions

Each library contains a termination function, which is normally the last function called. Functions called after the termination function will fail.

Typically the termination function takes a single parameter:

- The control block address returned from the initialization function

Message Callback functions

The server may supply the library with a function pointer that can be called to post messages back to the server log file. The callback function takes a single null-terminated string as its only parameter. There is no return code.

Control Blocks

The control block should be allocated in the initialization function. The address of the control block will then be passed back to all subsequent functions. Any data that needs to be shared between function calls should be placed in the control block. It's the responsibility of the termination function to delete the control block.

Each library defines the structure of its own control block.

Dope Vectors

Many of the libraries need the ability to call functions in other libraries that Docusave Server has loaded. Docusave Server uses a structure we're calling a "dope vector" to encapsulate the pertinent information about these libraries so that it can pass them to other libraries as a single data structure. The dope vector structure is defined in "dsapidv.h" and contains:

- The library handle
- The initialized control block
- The addresses of all the exported functions

Return Codes

Every library function returns a four-byte integer indicating the result of the function's execution. The valid return codes for each library are defined in the header file associated with that library. Some of the common return codes:

- **DS*_OK (0).** The function completed successfully.
- **DS*_REQUEST_FAILED (12).** The function did not complete successfully.
- **DS*_INVALID_PARM (16).** One or more of the parameters appears to be invalid. Examples: an invalid parameter address or a character string that is longer or shorter than expected.
- **DS*INVALID_CB (32).** The control block supplied to the function does not appear to be the same control block that was allocated in the initialization function.

Writing Job Recognition Libraries

Introduction

The Job Recognition Library identifies the job type when the job is pulled from the input queue. The Library returns a 3 character string that uniquely identifies the contents of the job, which Docusave Server uses as suffix to construct the name of the job processing libraries.

Depending on the job type coming in to the system, the job recognition library selects the appropriate job and document processing library to process the job.

Job Recognition Library

A working example of a recognition library is distributed as DSJBTXYZ.CPP.

Calling Sequence

The typical calling sequence for a recognition library is:

DSJOBTYP_init()
DSJOBTYP_setMsgCallback()
    DSJOBTYP_getType()
DSJOBTYP_terminate()

The initialization function (DSJOBTYP_init) is called one time before job servicing begins. The processing function (DSJOBTYP_getType) is called once for each job read from the input queue. Before Docusave Server is shut down the termination function (DSJOBTYP_terminate) is called.

Notes

The processing function (DSJOBTYP_getType) is given the control block from the initialization function, the name of the file containing the contents of the input job, and the job description supplied by the user when the job was added to the queue. The function passes back a unique three-character string (null-terminated) to identify the contents of the job. It also passes back a three-character string to indicate the type of processing that should be performed on this job. A null-string for the processing type indicates that the default processing for the job type should be performed.

API Description in C

```
/* Initialization functions */

DSJOBTYP_API CALL_TYPE  DSJOBTYP_init( DSJOBTYP *  cb,                 /* in/out */

                                       long        interfaceVersion );  /* in     */

DSJOBTYP_API CALL_TYPE  DSJOBTYP_setMsgCallback( DSJOBTYP  cb,          /* in     */
```

```
                                                  void *    pfn );       /* in     */


/* Recognition Functions */
DSJOBTYP_API CALL_TYPE  DSJOBTYP_getType( DSJOBTYP  cb,                 /* in     */
                                          char *    fileName,           /* in     */
                                          char *    description,        /* in     */
                                          char      jobType[4],         /* out    */
                                          char      processType[4] );   /* out    */
/* Termination function */
DSJOBTYP_API CALL_TYPE  DSJOBTYP_terminate( DSJOBTYP  cb );             /* in     */
```

Writing Job and Document Processing Libraries

Introduction

The Job recognition library ascertains the type of job in the queue, and thereby determines both the job processing and document processing libraries that will be used. The mechanism for automatically selecting the correct processing libraries is a *library naming convention*.

The library naming convention constructs processing library names from the job and document types (determined by the recognition step). If a pre-, post-, or processing library is present, and its name matches the naming convention for the current job, it will be invoked automatically at the appropriate time.

Job and document processing actions are implemented through separate libraries to allow a high degree of flexibility in controlling how processing occurs, and to provide custom behavior by using substitute or optional libraries.

Job processing refers to operations that happen once per job, where a job is a single entry in the input queue.

Document processing refers to operations that happen for each document in a job – potentially many times during one job. (A Job typically contains many documents.)

Job processing always use at least one library. The job processing library, may in turn, use one document processing library (for a total of 2 libraries). However, both job and document processing can also include optional pre-processing and post-processing libraries to bring the total number of processing libraries to 6. (Often pre- and post- processing libraries are not needed, and may not even be installed.)

Not surprisingly, the pre-processor library runs immediately before the corresponding job or document processor, and the post-processor runs immediately after.

The only purpose for using pre- and post- processing libraries is to modify (or supplement) the behavior of the main processing library. This makes it possible to use the standard processing libraries in non-standard situations.

For example, if you wish to validate index data in a database lookup before the standard document processor converts and files it, you might write a document pre-process library to validate the index data.

If you wanted to modify the result code assigned under a certain error condition, you might write a document post-processor library to test for that condition and modify the result code.

A custom job post-processor library could be used to provide additional notification (e.g., send an email or advance a workflow) when a job completes.

Pre- and Post- processing libraries can be simple or complex and are entirely optional.

In addition to pre- and post- processing, custom processing libraries can be used to create new functions and operations (e.g., convert print images to a different output type such as PDF). Custom processing libraries may or may not use pre- or post- processing libraries as well.

Job Processing Common Elements

Job and document pre-, post-, and processing libraries have a common calling sequence.

Calling Sequence

The typical calling sequence for job and document pre-, post-, and processing libraries is:

```
DSP*_init()
DSP*_set*()
    DSP*_process()
DSP*_terminate()
```

The initialization function is called exactly once immediately after the library is loaded into memory. One or more of the “set” functions may be called after the library is initialized. The processing function is called once for each job or document. Before Docusave Server unloads the library the termination function is called.

Custom Job Pre-Processing Libraries

A working example of a job pre-processing library is distributed as DSPRJXYZ.CPP.

Notes

DSPREPJ_preprocess()

The job pre-processing function is supplied with two file names: the name of the file containing the original job from the input queue, and the name of a file that may be used to place an altered version of this input job. The process type (returned from the recognition library) is also supplied to this function and may be altered in-place if desired. This function should return DSPREPJ_USE_ORIGINAL if the original input file should be passed to the job processing step or DSPREPJ_OK if the altered job should be used instead.

API Description in C

```
/* Initialization functions */
DSPREPJ_API CALL_TYPE      DSPREPJ_init( DSPREPJ *  cb,                /* in/out */
                                         long       interfaceVersion ); /* in     */
DSPREPJ_API CALL_TYPE      DSPREPJ_setImgSys( DSPREPJ            cb,    /* in     */
                                              DSapiDopeVector *  pdv ); /* in     */
DSPREPJ_API CALL_TYPE      DSPREPJ_setMsgCallback( DSPREPJ  cb,         /* in     */
                                                   void *   pfn );      /* in     */

/* Processing function */
DSPREPJ_API CALL_TYPE      DSPREPJ_preprocess( DSPREPJ  cb,             /* in     */
                                               char *   fileIn,         /* in     */
                                               char *   fileOut,        /* in     */
                                               char *   processType );  /* in/out */
/* Termination function */
DSPREPJ_API CALL_TYPE      DSPREPJ_terminate( DSPREPJ  cb );            /* in     */
```

Custom Job Processing Libraries

A working example of a job processing library is distributed as DSPJBXYZ.CPP.

Notes

DSPROCJ_process()

The job processing function is supplied with the name of the file containing the job and a character string that indicates the type of processing to be performed. (This character string may be null, which indicates the default processing type.) Any meaning associated with the processing type is determined in this function.

At the discretion of this function, document processing libraries (document pre-, post-, and processing) may be called to handle the individual documents inside the job. This may or may not be applicable depending on the job type. For information on Document Processing go to “Custom Document Pre-Processing Libraries” on page 29”

This function would normally call the beginJob() and endJob() functions of the Imaging library. If no document processing libraries are used (i.e. all processing takes place in this function) then this function would also call the other Imaging library functions required to store each of the documents in the job. See “Writing Imaging Libraries” on page 33 for more information.

API Description in C

```
/* Initialization functions */
DSPROCJ_API CALL_TYPE      DSPROCJ_init( DSPROCJ *  cb,                          /*
in/out */
                                      long      interfaceVersion );           /*
in     */
DSPROCJ_API CALL_TYPE      DSPROCJ_setCompression( DSPROCJ            cb,        /*
in     */
                                                   DSapiDopeVector *  pdv );      /*
in     */
DSPROCJ_API CALL_TYPE      DSPROCJ_setImgSys( DSPROCJ            cb,            /*
in     */
                                           DSapiDopeVector *  pdv );          /*
in     */
DSPROCJ_API CALL_TYPE      DSPROCJ_setMsgCallback( DSPROCJ  cb,                  /*
in     */
```

```
                                                    void *  pfn );                  /*
in     */
DSPROCJ_API CALL_TYPE      DSPROCJ_setup( void *  parentWindow );                  /*
in     */

/* Processing */
DSPROCJ_API CALL_TYPE      DSPROCJ_process( DSPROCJ  cb,                           /*
in     */
                                            char *  fileName,                       /*
in     */
                                            char *  processType );                  /*
in     */
/* Termination function */
DSPROCJ_API CALL_TYPE      DSPROCJ_terminate( DSPROCJ  cb );                       /*
in     */
```

Custom Job Post-Processing Libraries

A working example of a job post-processing library is distributed as DSPOJXYZ.CPP.

Notes

DSPOSTJ_postprocess()

The job post-processing function is supplied with the name of the file and character string passed to the DSPROCJ_process() function. It is also given the address of a four-byte integer which contains the return code from the DSPROCJ_process() function. This function may adjust the return code if necessary.

API Description in C

```
/* Initialization functions */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_identity( char  identity[ 64 ] );    /* out    */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_init( DSPOSTJ *  cb,                 /* in/out */
                                         long       interfaceVersion ); /* in     */
```

```
DSPOSTJ_API CALL_TYPE      DSPOSTJ_setImgSys( DSPOSTJ           cb,    /* in     */
                                              DSapiDopeVector *  pdv ); /* in     */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_setMsgCallback( DSPOSTJ  cb,        /* in     */
                                                   void *   pfn );     /* in     */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_setup( void *  parentWindow );      /* in     */

/* Processing function */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_postprocess( DSPOSTJ  cb,           /* in     */
                                               char *   fileIn,        /* in     */
                                                char *   processType,  /* in     */
                                                long *   rc );         /* in/out */

/* Termination function */
DSPOSTJ_API CALL_TYPE      DSPOSTJ_terminate( DSPOSTJ  cb );           /* in     */
```

Custom Document Pre-Processing Libraries

A working example of a document pre-processing library is distributed as DSPRDXYZ.CPP.

Notes

DSPREPD_preprocess()

The document pre-processing function is supplied with two file names: the name of the file containing the original document from the input job, and the name of a file that may be used to place an altered version of this input document. The process type (returned from the recognition library) is also supplied to this function and may be altered in-place if desired. This function should return DSPREPD_USE_ORIGINAL if the original input file should be passed to the document processing step or DSPREPD_OK if the altered document should be used instead.

API Description in C

```
/* Initialization functions */
DSPREPD_API CALL_TYPE      DSPREPD_init( DSPREPD *  cb,                /* in/out */
                                         long       interfaceVersion ); /* in     */
DSPREPD_API CALL_TYPE     DSPREPD_setImgSys( DSPREPD          cb,    /* in    */
                                             DSapiDopeVector *  pdv ); /* in     */
DSPREPD_API CALL_TYPE      DSPREPD_setMsgCallback( DSPREPD  cb,        /* in     */
                                                   void *   pfn );     /* in     */

/* Processing function */
DSPREPD_API CALL_TYPE      DSPREPD_preprocess( DSPREPD  cb,            /* in     */
                                               char *   fileIn,        /* in     */
                                               char *   fileOut,       /* in     */
                                               char *   processType ); /* in/out */
/* Termination function */
DSPREPD_API CALL_TYPE      DSPREPD_terminate( DSPREPD  cb );           /* in     */
```

Custom Document Processing Libraries

A working example of a document processing library is distributed as DSPDCXYZ.CPP.

Notes

DSPROCD_process()

The document processing function is supplied with the name of the file containing the document and a character string that indicates the type of processing to be performed. (This character string may be null, which indicates the default processing type.) Any meaning associated with the processing type is determined in this function.

This function would normally call the setDocType(), setIndex(), and setPageCount() functions of the Imaging library followed by the beginDoc(),

appendPage(), and endDoc() Imaging library functions. See “Writing Imaging Libraries” on page 33 for more information.

API Description in C

```
/* Initialization functions */
DSPROCD_API CALL_TYPE      DSPROCD_init( DSPROCD *  cb,                          /*
in/out */
                                    long     interfaceVersion );         /* in
*/
DSPROCD_API CALL_TYPE     DSPROCD_setCompression( DSPROCD           cb,     /* in
*/
                                              DSapiDopeVector *  pdv );    /* in
*/
DSPROCD_API CALL_TYPE     DSPROCD_setImgSys( DSPROCD           cb,          /* in
*/
                                         DSapiDopeVector *  pdv );        /* in
*/
DSPROCD_API CALL_TYPE     DSPROCD_setMsgCallback( DSPROCD  cb,             /* in
*/
                                          void *   pfn );             /* in
*/
DSPROCD_API CALL_TYPE     DSPROCD_setup( void *  parentWindow );           /* in
*/

/* Processing */
DSPROCD_API CALL_TYPE     DSPROCD_process( DSPROCD  cb,                    /* in
*/
                                      char *  fileName,                   /* in
*/
                                      char *  processType );              /* in
*/

/* Termination function */
DSPROCD_API CALL_TYPE     DSPROCD_terminate( DSPROCD  cb );                /* in
*/
```

Custom Document Post-Processing Libraries

A working example of a document post-processing library is distributed as DSPODXYZ.CPP.

Notes

DSPOSTD_postprocess()

The document post-processing function is supplied with the name of the file and character string passed to the DSPROCD_process() function. It is also given the address of a four-byte integer which contains the return code from the DSPROCD_process() function. This function may adjust the return code if necessary.

API Description in C

```
/* Initialization functions */
DSPOSTD_API CALL_TYPE      DSPOSTD_init( DSPOSTD *  cb,                          /*
in/out */
                                    long      interfaceVersion );         /* in
*/
DSPOSTD_API CALL_TYPE     DSPOSTD_setImgSys( DSPOSTD           cb,          /* in
*/
                                         DSapiDopeVector *  pdv );        /* in
*/
DSPOSTD_API CALL_TYPE     DSPOSTD_setMsgCallback( DSPOSTD  cb,             /* in
*/
                                            void *   pfn );             /* in
*/
DSPOSTD_API CALL_TYPE     DSPOSTD_setup( void *  parentWindow );           /* in
*/

/* Processing */
DSPOSTD_API CALL_TYPE     DSPOSTD_postprocess( DSPOSTD  cb,                /* in
*/
                                       char *   fileName,                /* in
*/
                                       char *   processType,             /* in
*/
                                           long *   rc );                /*
in/out */

/* Termination function */
DSPOSTD_API CALL_TYPE     DSPOSTD_terminate( DSPOSTD  cb );                /* in
*/
```

Writing Imaging Libraries

Introduction

Imaging Libraries are used to store and retrieve documents to and from an imaging system. Each imaging system has its own library.

Custom Imaging Libraries

A working example of an imaging library is distributed as DSISYSU.CPP.

Calling Sequence

The typical calling sequence to store a document in an imaging library is:

```
DSIMGSYS_init()
DSIMGSYS_setMsgCallback()
DSIMGSYS_isConnected()
        DSIMGSYS_beginJob()
                DSIMGSYS_setDocType()
                DSIMGSYS_setPageCount()
                DSIMGSYS_setIndex()
                DSIMGSYS_beginDoc()
                        DSIMGSYS_appendFilePage() or
```

DSIMGSYS_appendMemPage()
DSIMGSYS_endDoc()
DSIMGSYS_endJob()
DSIMGSYS_terminate()

The typical calling sequence to retrieve a document from an imaging library is:

DSIMGSYS_init()
DSIMGSYS_setMsgCallback()
DSIMGSYS_isConnected()
DSIMGSYS_fetchDoc()
DSIMGSYS_fetchPage()
DSIMGSYS_terminate()

Notes

Any type of login to the imaging system should be done when DSIMGSYS_init() is called. If the login fails then subsequent calls to DSIMGSYS_ping() and DSIMGSYS_isConnected() should also fail.

DSIMGSYS_ping() will be called at intervals to keep the imaging system connection from timing out.

DSIMGSYS_isConnected() will normally be called immediately before a job is processed to verify that a valid connection still exists.

API Description in C

```
/* Initialization functions */
DSIMGSYS_API CALL_TYPE      DSIMGSYS_init( DSIMGSYS *  cb,                      /*
in/out */
                                    long       interfaceVersion,       /* in
*/
```

```
                                                char        selfDescription[32] );   /*
out    */
DSIMGSYS_API CALL_TYPE     DSIMGSYS_isConnected( DSIMGSYS  cb );            /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_setMsgCallback( DSIMGSYS  cb,           /* in
*/
                                                    void *    pfn );          /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_setup( void *  parentWindow );          /* in
*/

/* Processing functions */
DSIMGSYS_API CALL_TYPE     DSIMGSYS_beginJob( DSIMGSYS  cb );               /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_beginDoc( DSIMGSYS  cb );               /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_setDocType( DSIMGSYS  cb,               /* in
*/
                                                long      docType,           /* in
*/
                                                char *    fileExt );         /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_setIndex( DSIMGSYS  cb,                 /* in
*/
                                               char *    indexString );       /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_setPageCount( DSIMGSYS  cb,             /* in
*/
                                                  long      nPages );        /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_appendFilePage( DSIMGSYS  cb,           /* in
*/
                                                    char *    fileName );    /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_appendMemPage( DSIMGSYS  cb,            /* in
*/
                                                   void *    pData,          /* in
*/
                                                   long      nBytes );       /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_endDoc( DSIMGSYS  cb );                 /* in
*/
DSIMGSYS_API CALL_TYPE     DSIMGSYS_endJob( DSIMGSYS  cb );                 /* in
*/

DSIMGSYS_API CALL_TYPE     DSIMGSYS_fetchDoc( DSIMGSYS  cb,                 /* in
*/
                                              char *    url,                 /* in
*/
                                                  long *    totalPages );          /*
out    */
DSIMGSYS_API CALL_TYPE     DSIMGSYS_fetchPage( DSIMGSYS  cb,                /* in
*/
                                               long      n,                  /* in
*/
                                               char *    fileName );         /* in
*/
```

```
DSIMGSYS_API CALL_TYPE     DSIMGSYS_resourceExists( DSIMGSYS  cb,           /* in
*/
                                                    char *    resName,       /* in
*/
                                                    long      resType,       /* in
*/
                                                   char *    yyyymmddhhmmss );/* in
*/

/* Termination function */
DSIMGSYS_API CALL_TYPE     DSIMGSYS_terminate( DSIMGSYS  cb );              /* in
*/
```